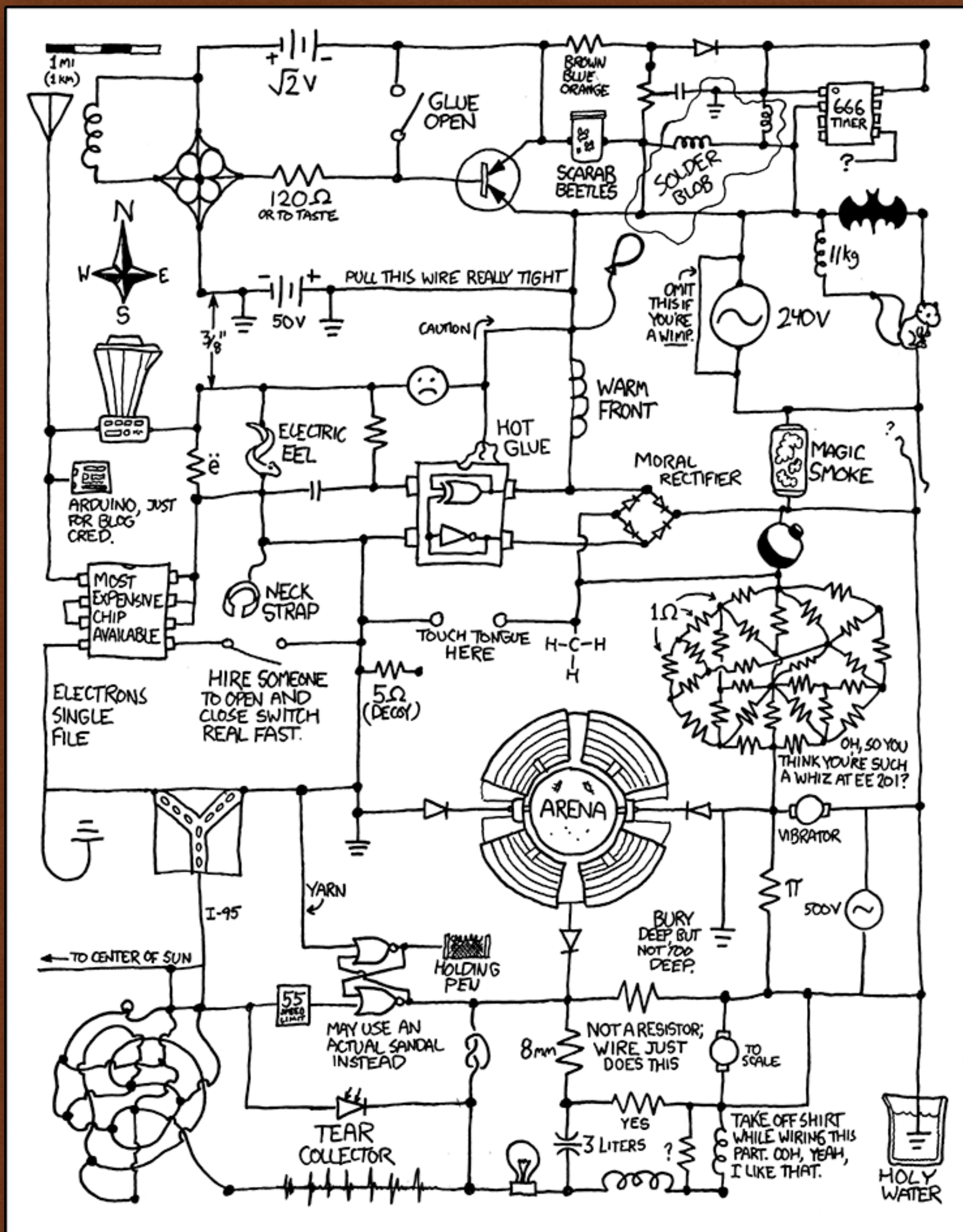


THE BUILDING BLOCKS: SEQUENTIAL CIRCUITS

https://xkcd.com/730/ - find the latch

LEARNING OUTCOMES

- Selecting values from many (multiplexor)
- Using bits to choose what to do (decoder) The difference between combinational and sequential circuits
- Simple latches
- Implementing registers with simple latches

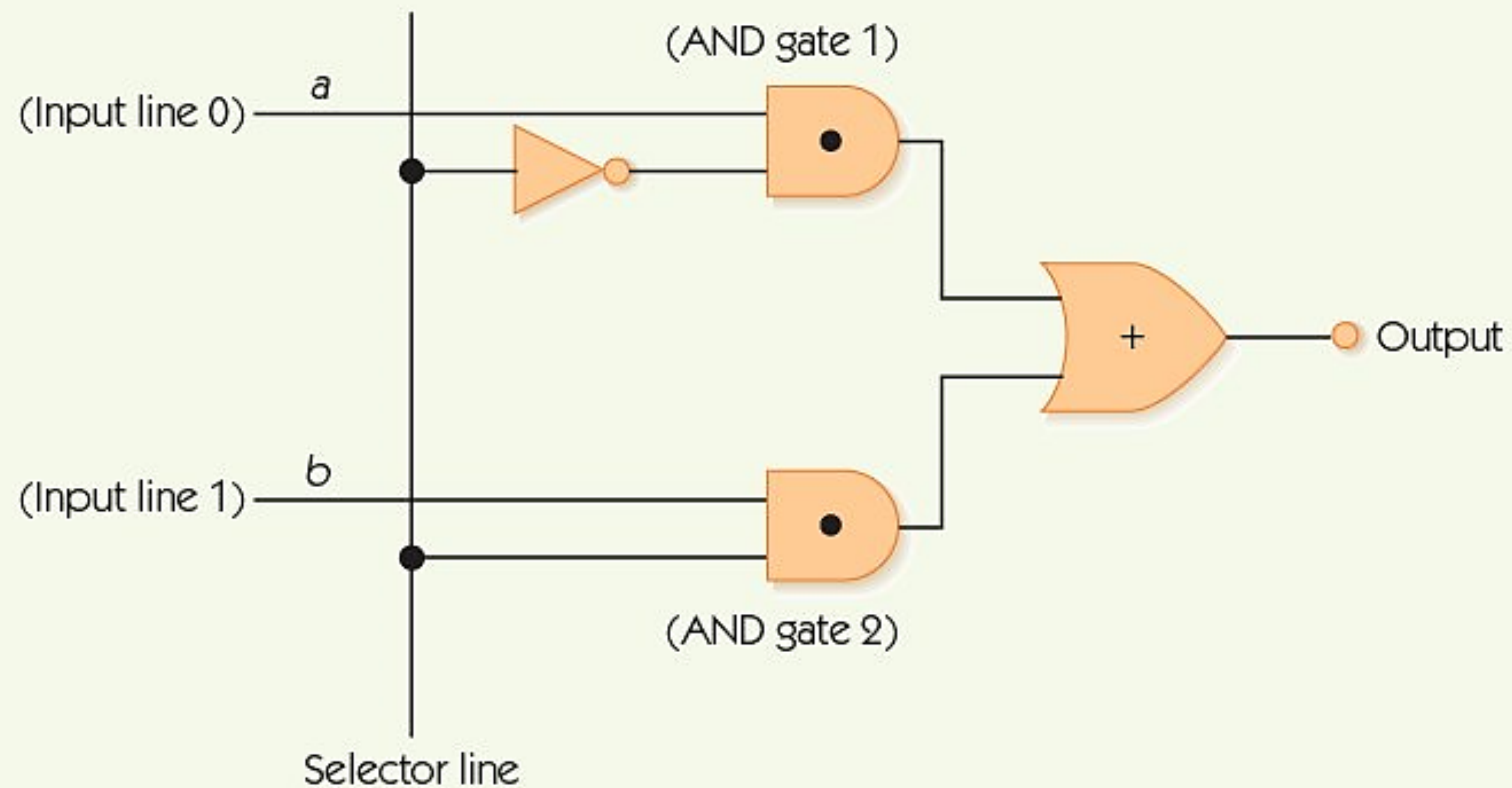
CONTROL CIRCUITS

- **Control circuits** make decisions, determine order of operations, select data values
- **Multiplexer** (**mux**) selects one from among many inputs
  - 2^n input lines e.g. 8
  - n selector lines e.g. 3
  - 1 output line - with the output being one of the inputs
- e.g. If we have 4 things to select from we need 2 selector lines
- Each input line corresponds to a unique pattern on selector lines
- That input value is passed to output

2 SOURCE MULTIPLEXOR

n = 1

FIGURE 4.34

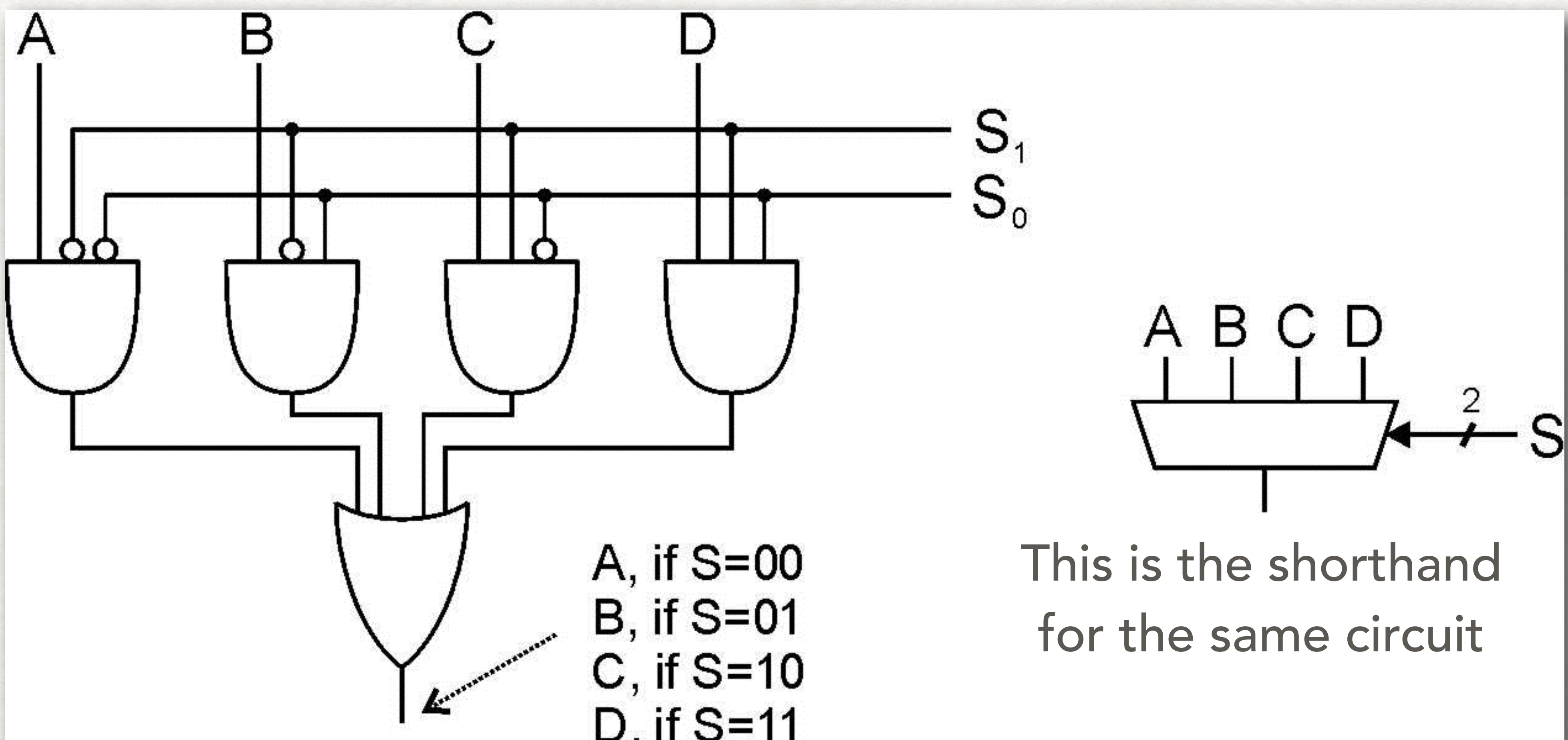


A two-input multiplexor circuit

4 SOURCE MULTIPLEXOR

n = 2

- Using the alternative way of drawing circuits it is easier to see how this multiplexor works.

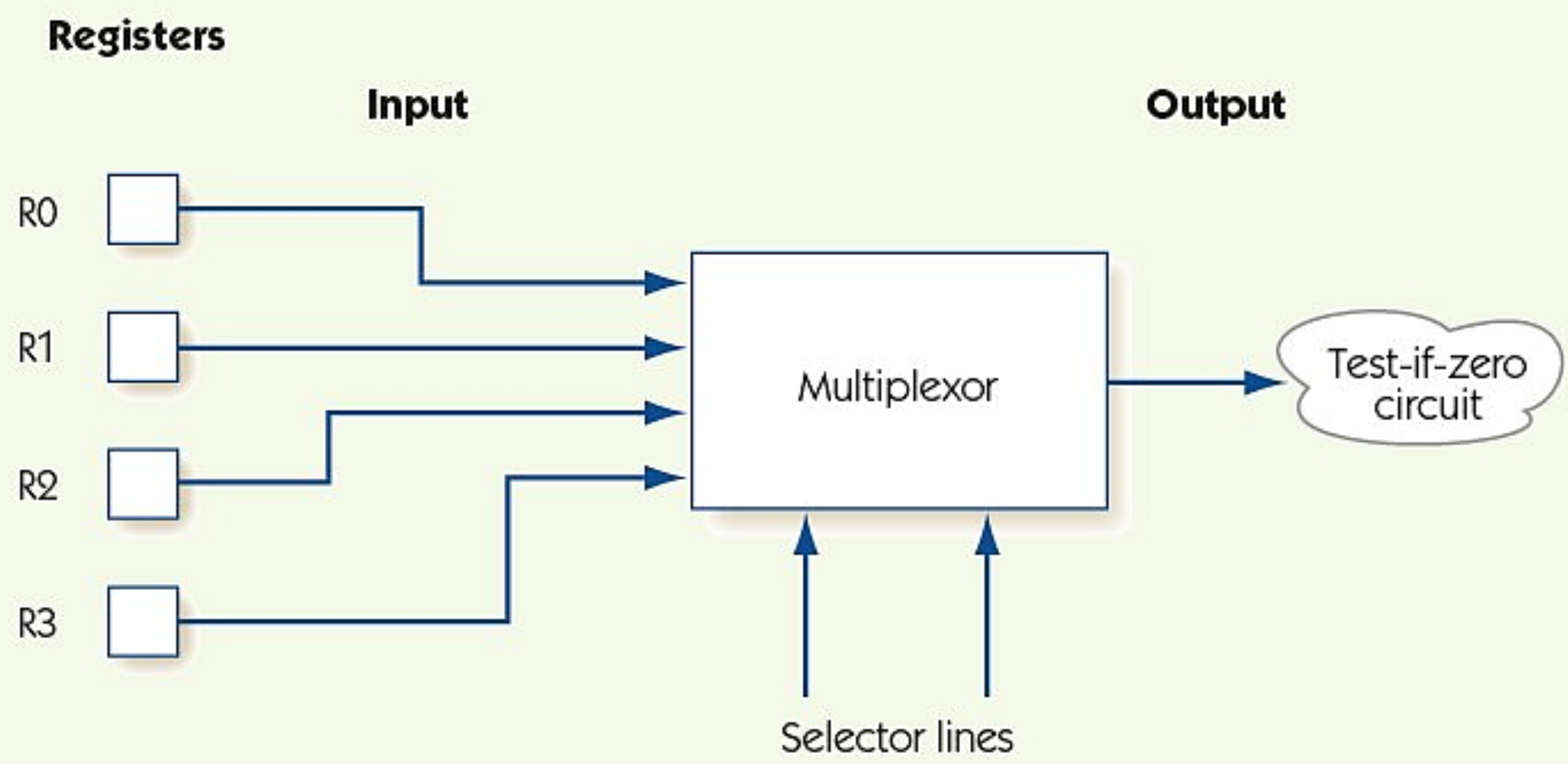


WHAT ABOUT N BIT VALUES?

- The previous 2 slides show 1 of 2 bits or 1 of 4 bits being selected by the multiplexor
- We commonly want to choose 1 of a group of n-bits (e.g.16 bit values $a_{15}a_{14}...a_2a_1a_0$)
  - to do this we need n multiplexors all connected to the same selector lines
  - each multiplexor passes through one of the bits a_i from the selected value

CHOOSING ONE OF FOUR

FIGURE 4.38



Example of the use of a multiplexor circuit

EXAMPLES OF USING A MULTIPLEXOR

- In a CPU (the processor) multiplexors are used to choose from a variety of inputs e.g. we may have an adder which can add a value from a register or from memory, a multiplexor can choose which one.

- As we will see, CPUs have a special register - the program counter (PC) (which holds a number) - to point to the next instruction to be executed. Normally we add one to the PC to move on to the next instruction. Sometimes we add a different value; a multiplexor could be used to choose the value to add.
- Many data values flow into the multiplexer, only the selected one comes out

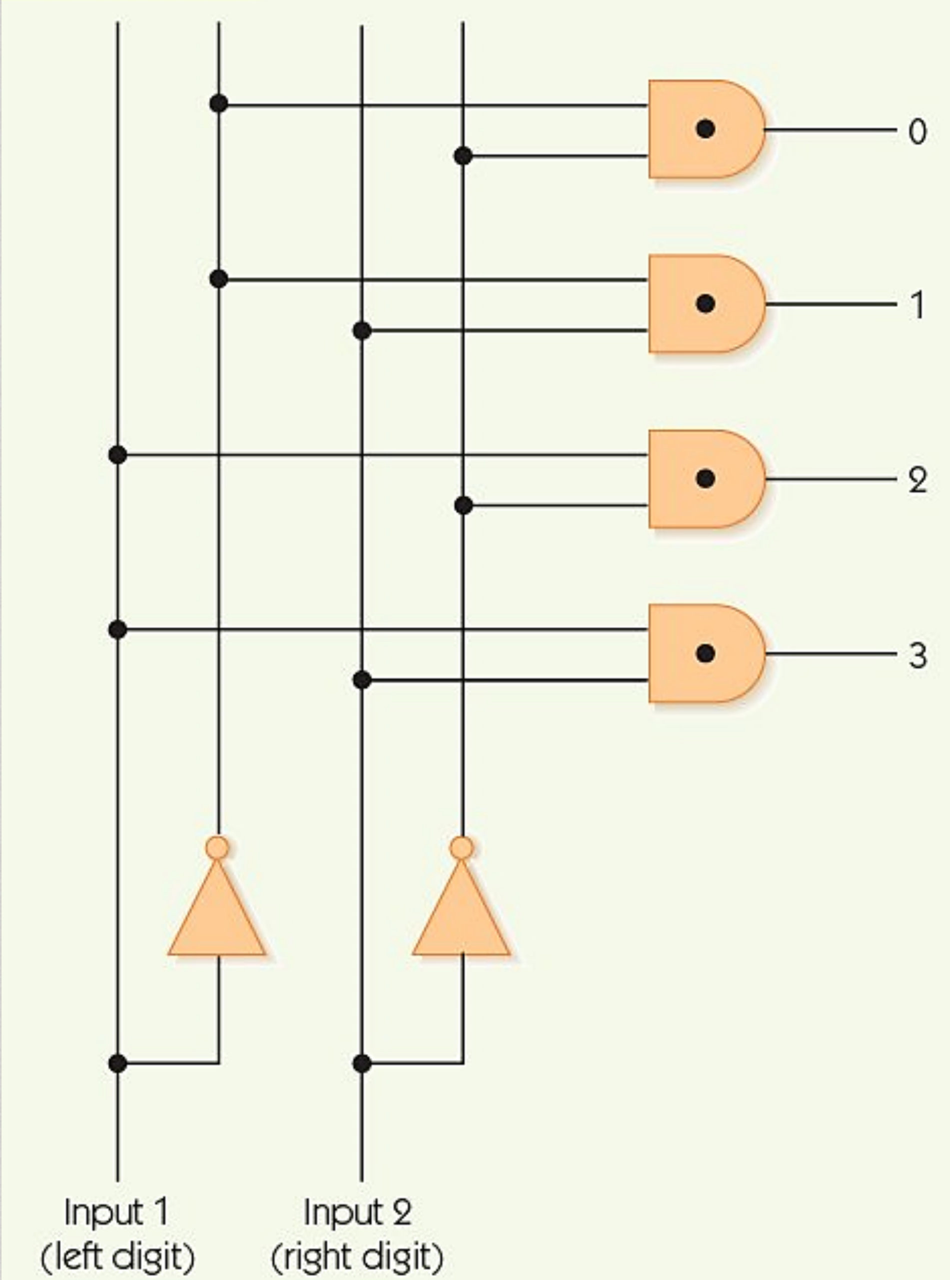
DECODER

- **Decoder** sends a signal out to only one output chosen by its input
    - n input lines e.g. 4
    - 2^n output lines e.g. 16
- Sort of like the inverse of a multiplexor, the value coming in on the n input lines chooses one of the 2^n output lines (this line has a value of 1)
- Each output line corresponds to a unique pattern on input lines. In the example a number from 0 to 15 (4 bits) indicates which of the 16 lines get switched on.
- Only the chosen output line produces 1, all others output 0

2 TO 4 DECODER

n = 2

FIGURE 4.36



A 2-to-4 decoder circuit

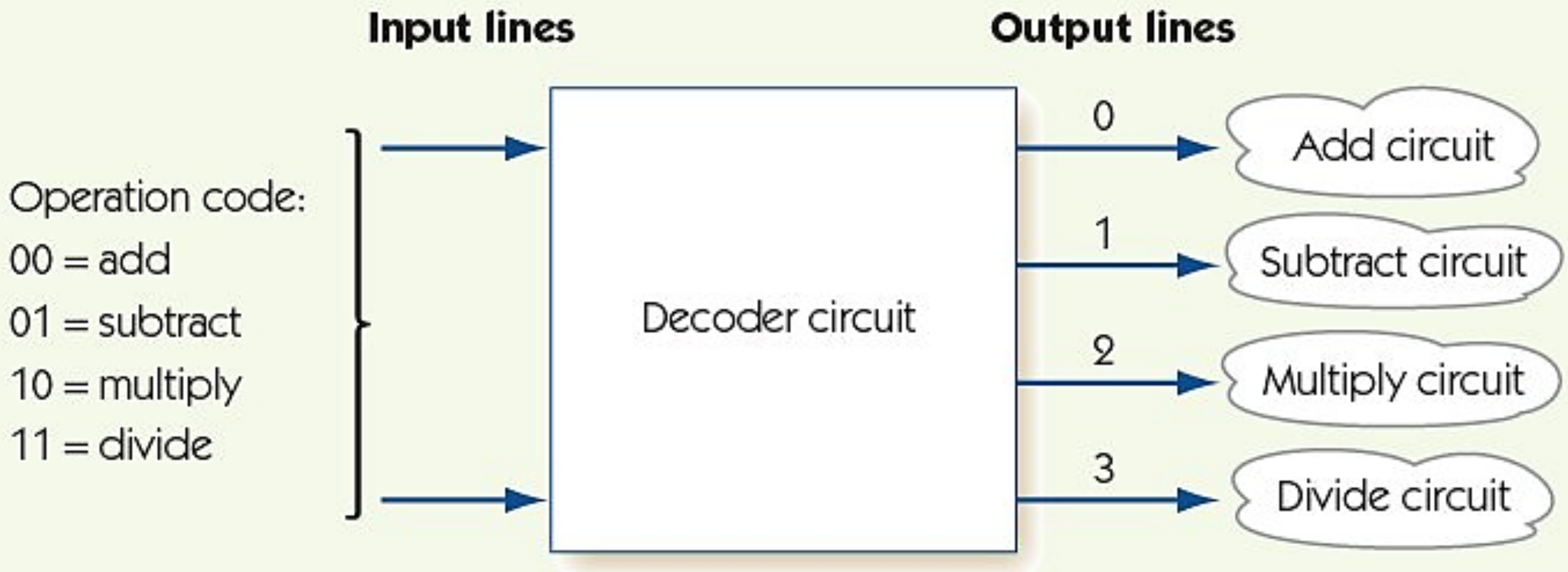
2 bits determine which of 4 output lines is active

EXAMPLES OF USING A DECODER

- Decoder circuit uses
  - To select a single arithmetic instruction, given a code for that instruction (e.g. + is 0, - is 1, × is 2, ÷ is 3)
  - Code activates one output line; that line activates corresponding arithmetic circuit
  - More generally we will see we need a way to do different things in our CPUs for different instructions, a decoder is needed to take the bits of an instruction and send signals to other parts of the processor particular to each instruction, e.g. LOAD or STORE or ADD …

TYPICAL DECODER

FIGURE 4.37



Example of the use of a decoder circuit

TWO TYPES OF CIRCUITS

- Combinational Logic Circuit
  - output depends only on the current inputs
  - stateless
  - all the circuits we have seen so far have been combinational
- Sequential Logic Circuit
  - output depends on the sequence of inputs (past and present)
  - stores information (state) from past inputs

COMBINATIONAL VS SEQUENTIAL

- Combinational Circuit
  - always gives the same output for a given set of inputs
  - example: adder always generates same sum and carry if given the same numbers to add regardless of previous inputs
- Sequential Circuit
  - stores information
  - output depends on stored information (state) plus input
  - so a given input might produce different outputs, depending on the stored information
  - example: ticket counter
    - advances when you push the button
    - output depends on previous state
  - useful for building “memory” elements and “state machines”

A SIMPLE LATCH

- A latch holds a value, e.g. the state usually stays either on or off and can be changed from one to the other when required
- Back to Minecraft - https://www.youtube.com/watch?v=KXScOWZNdBg and the rest of https://www.youtube.com/watch?v=OOWaYNf35X4 - 2 minutes 11
- To hold a value we need the circuit to feed back into itself

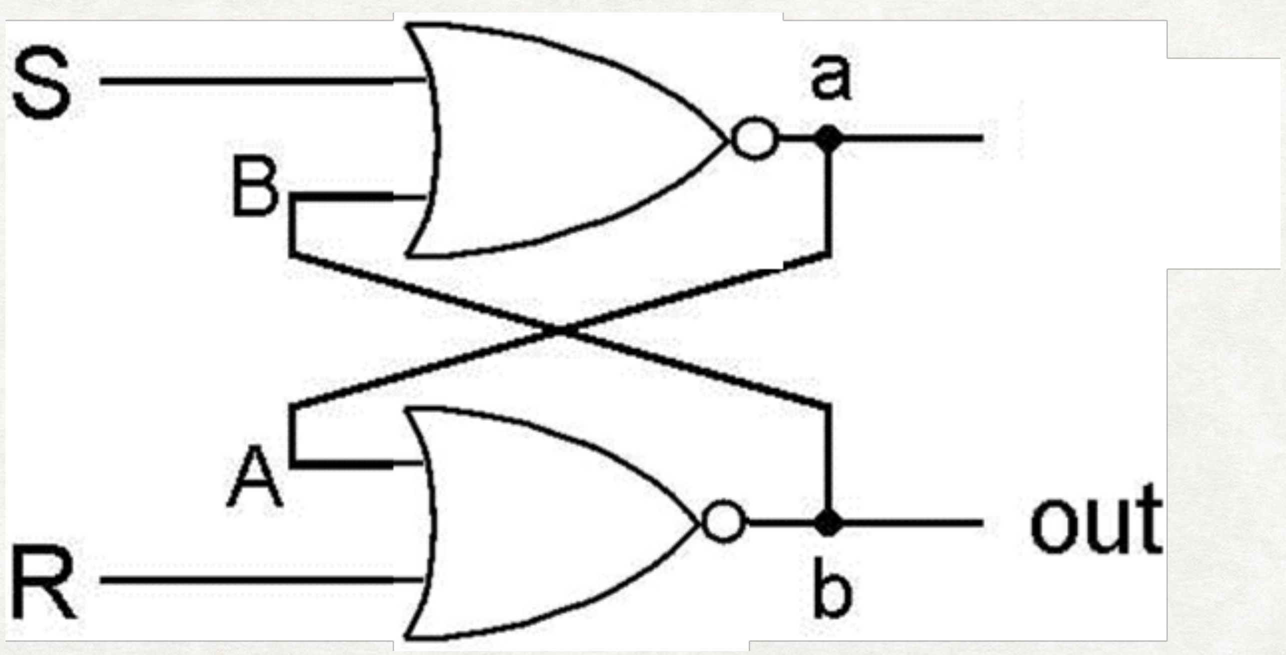
THE SR LATCH

- A Set Reset latch can be created using NOR or NAND gates we will only look at the NOR gate version.

- The S input means Set the latch i.e. make it 1

- The R input means Reset the latch i.e. make it 0

A	B	A nor B
0	0	1
0	1	0
1	0	0
1	1	0

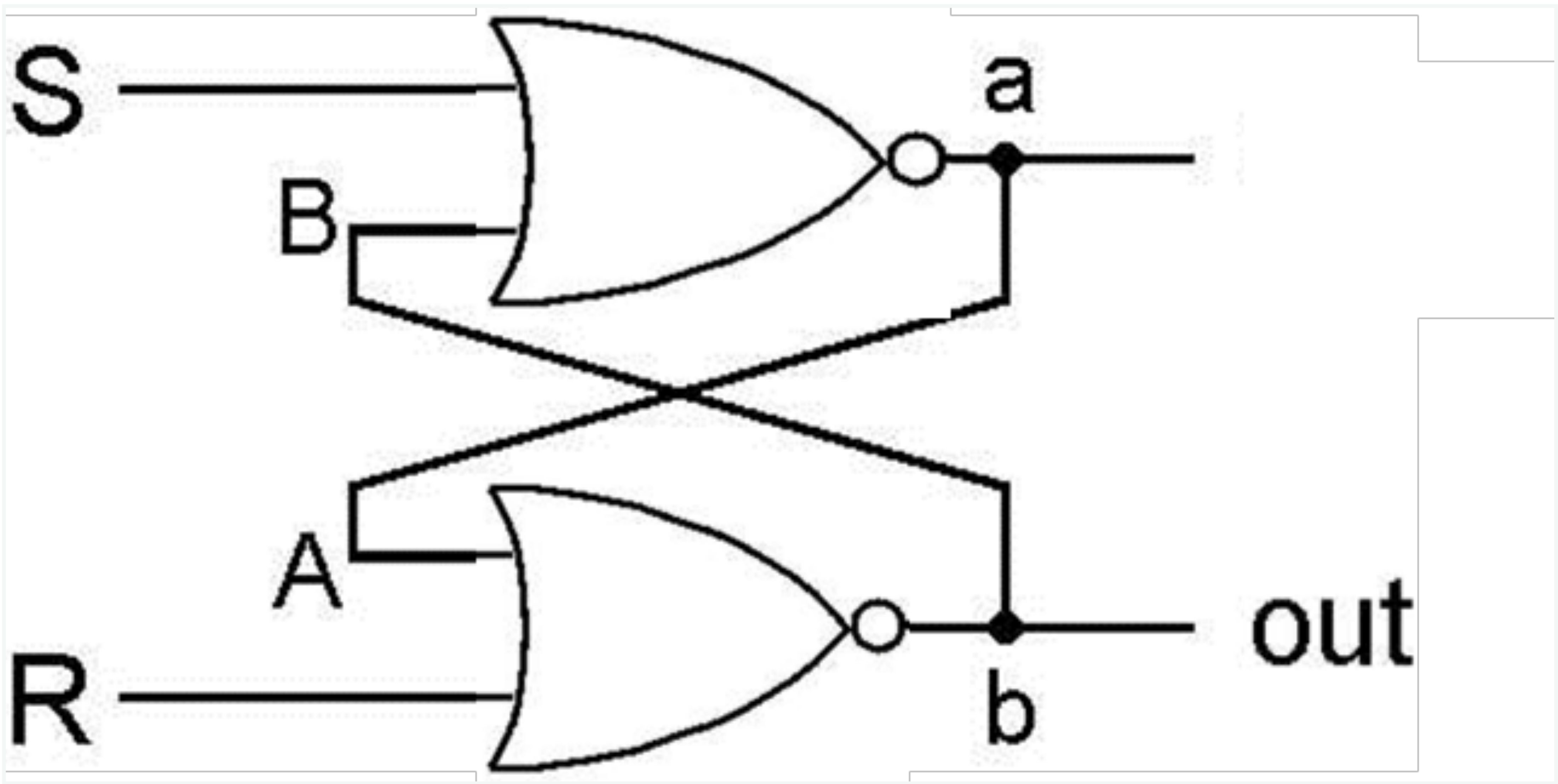


THE SR LATCH TABLE

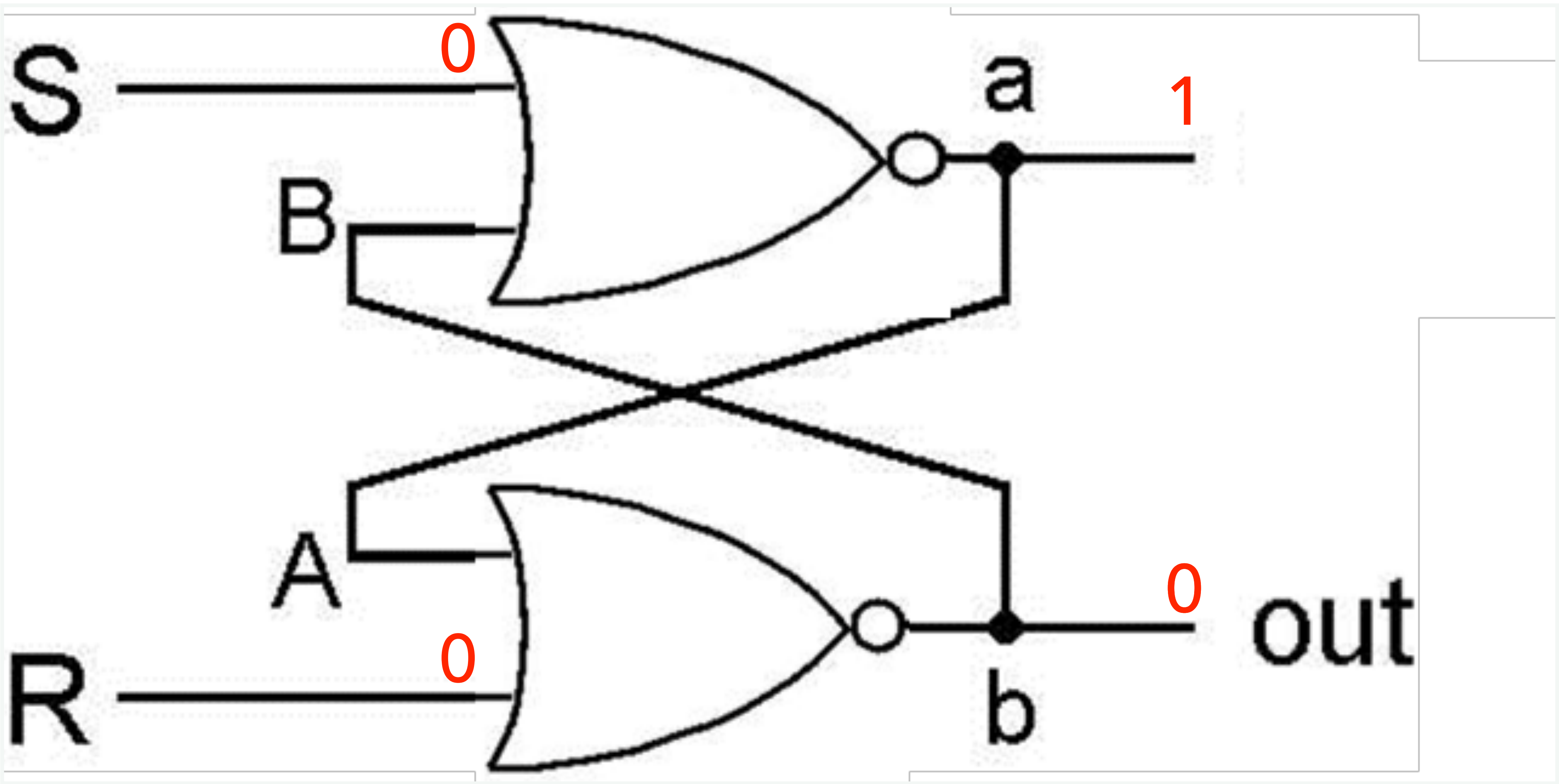
	S	R	existing out	resulting out
no change	0	0	0	0
Set	1	0	0	1
no change	1	0	1	1
Reset	0	1	1	0
no change	0	1	0	0
no change	0	0	1	1
DON'T	1	1	1	Aaargh!
DON'T	1	1	0	Aaargh!

After a set or reset the S or R reverts to zero, this keeps the value.
So S to one then back to zero makes the output one and
R to one then back to zero makes the output zero.

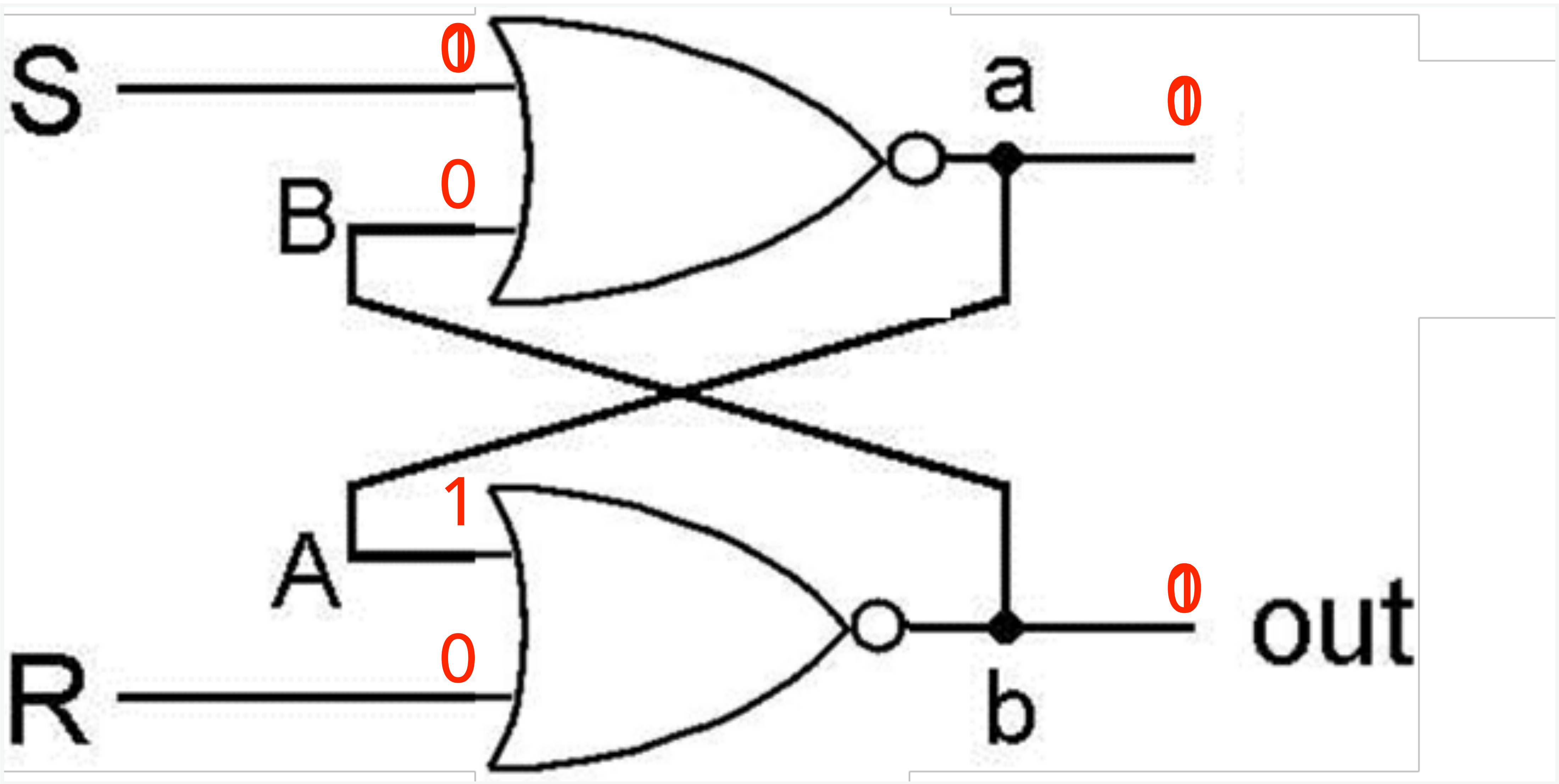
FILL IN THE VALUES AND SEE HOW THEY CHANGE



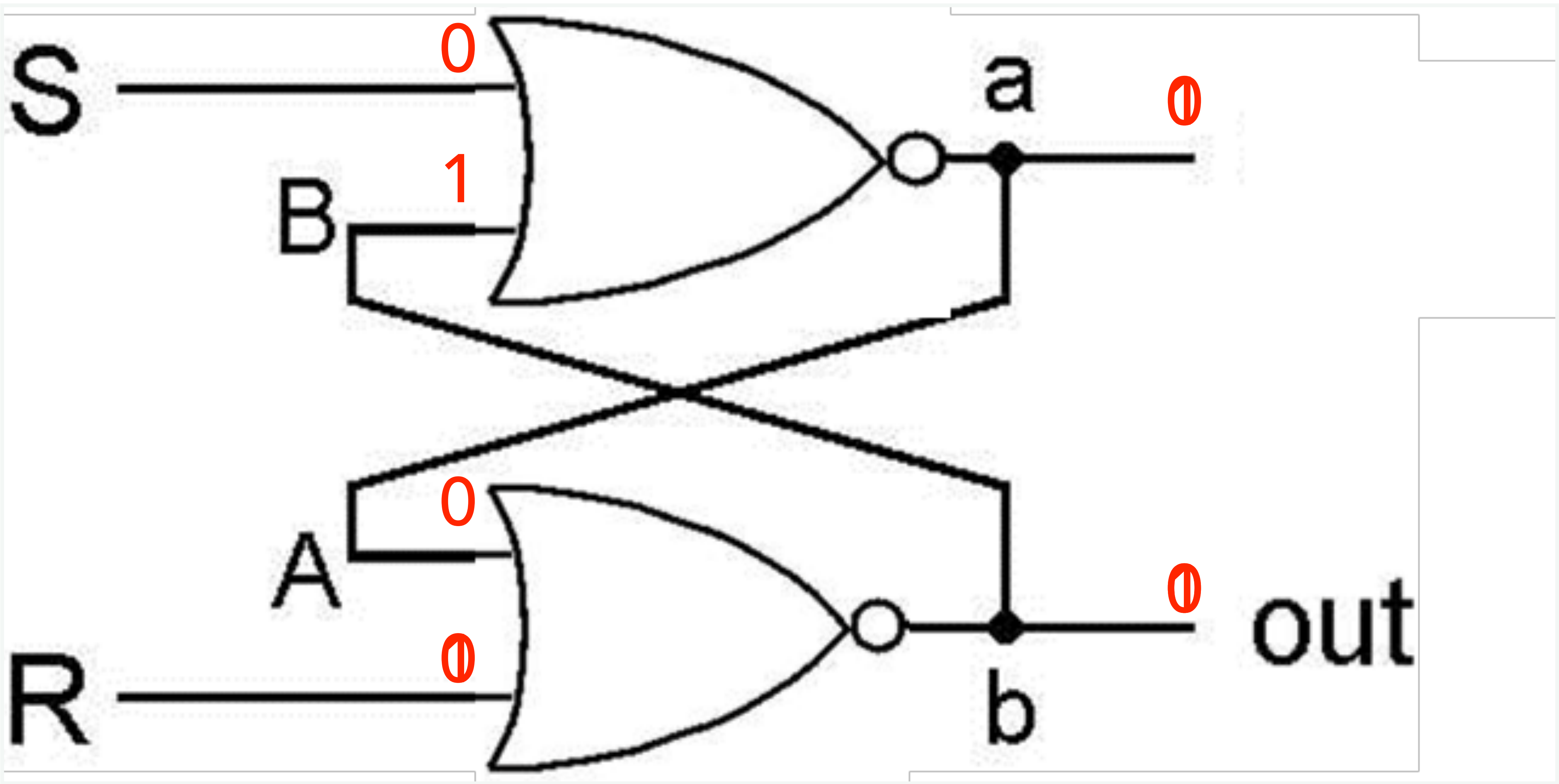
	S	R	existing out	resulting out
no change	0	0	0	0



	S	R	existing out	resulting out
Set	1	0	0	1

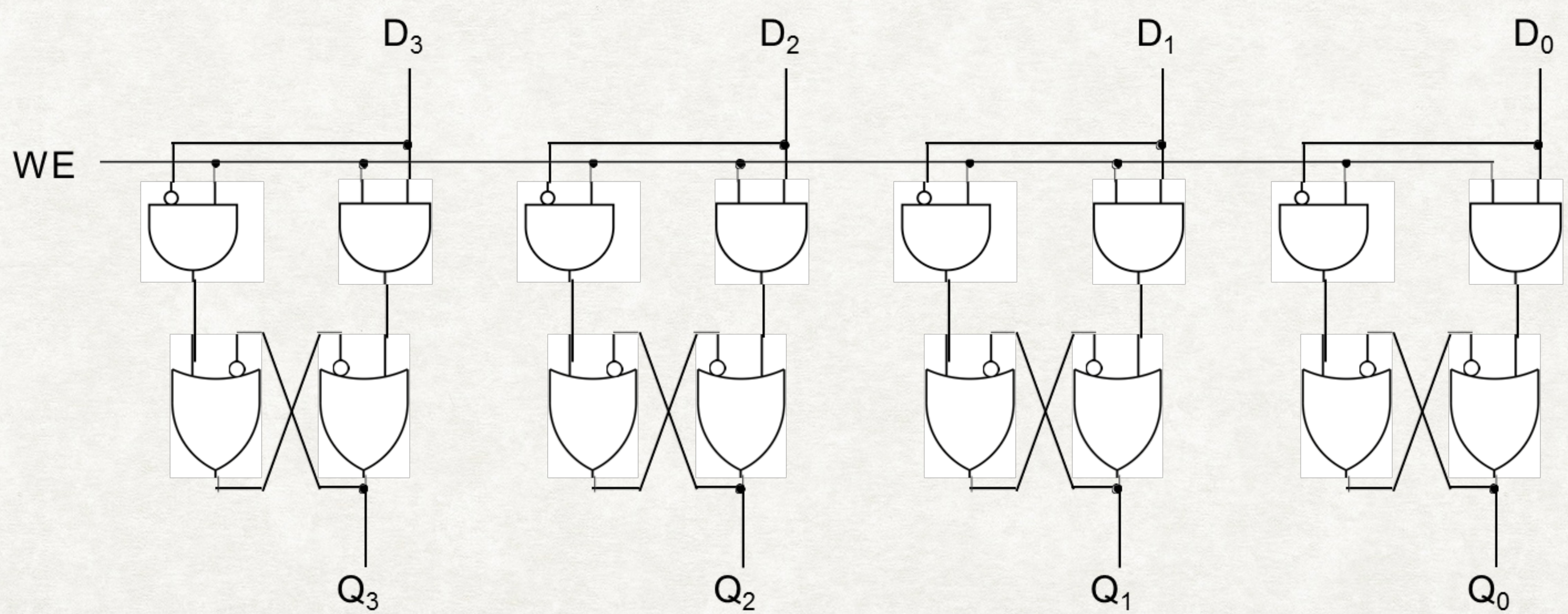


	S	R	existing out	resulting out
Reset	0	1	1	0



REGISTERS

- A register holds a number of bits - commonly the "word" size of the computer
- We can build a 4 bit register with 4 SR latches
- The latches here are made from OR gates, they are still SR latches
- If D_i is 1 then we store a 1, if 0 we store a 0
- The WE is Write Enable and we set the 4-bit value of the register only when this is 1



PUTTING IT TOGETHER

- We already knew how to encode numbers (and hence any data) into bits. We now know how to create n-bit registers e.g. 16-bit
- If we have two 16-bit registers we can compare corresponding bits from the two registers using the n-bit comparison circuit from the last lecture, i.e. we are comparing two 16-bit numbers
- Similarly we can add or subtract two 16-bit registers by using a 16-bit full adder from lecture 8
- So we can make a simple calculator - we are not going to consider multiplication and division
- The question is what else do we need to add in order to make a computer?